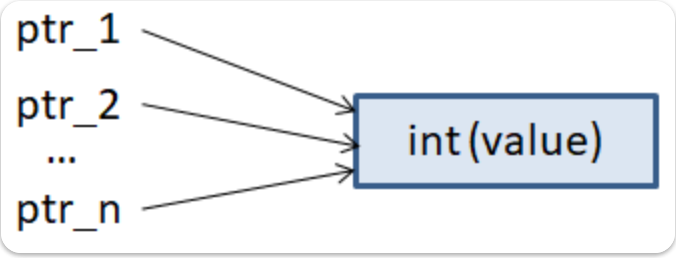
На предыдущем занятии мы с вами подробно разобрали smart-указатели типа unique_ptr, которые могут только в единственном числе ссылаться на выделенную область памяти и автоматически ее освобождать, значительно уменьшая риск утечки памяти. Однако в практике программирования чаще используются указатели вида shared_ptr, которые также автоматически освобождают неиспользуемую память, но в отличие от unique_ptr могут ссылаться на нее во множественном числе:



Указатели типа shared_ptr объявляются в программе по аналогии с указателями unique_ptr. Вначале также нужно подключить заголовок:

```
#include <memory>
```

После чего в пространстве имен std появится класс shared_ptr:

```
std::shared_ptr<int> ptr;
std::shared_ptr<int> ptr_2 {};
std::shared_ptr<int> ptr_3 {nullptr};
std::shared_ptr<int> ptr_4 {ptr};
```

Во всех случаях объявляются указатели со значением nullptr. Причем, последний указатель ptr_4 инициализирован адресом указателя ptr. Как мы помним, с типом unique_ptr такая конструкция приводила к ошибке на этапе компиляции. Здесь же никаких ошибок нет, т.к. shared_ptr предполагает множество разных smart-указателей на одну и ту же область памяти.

Если нам нужно сразу в инициализаторе указателя типа shared_ptr выделить некоторую область памяти, то это следует делать с помощью функции make_shared():

```
std::shared_ptr<int> ptr {std::make_shared<int>(3)};
```

В результате ptr будет ссылаться на область памяти для хранения целочисленного значения 3 типа int.

Пользоваться указателем ptr можно абсолютно так же, как и обычным указателем на тип int. Например:

```
ptr_2 = ptr; // присвоение адреса указателя ptr указателю ptr_2
*ptr = 10;  // запись числа 10 по адресу указателя ptr
cout << ptr_2 << *ptr_2 << endl;  // чтение и вывод в консоль значения по указателю ptr_2
```

Обратите внимание, что операция взятия адреса указателя shared_ptr делается просто по его имени (без вызова метода get).

А вот выполнять операции адресной арифметики с такими указателями не получится. При выполнении следующих команд получим ошибки на этапе компиляции:

```
ptr_3 += 10;
auto res = ptr_3 - ptr_4;
```

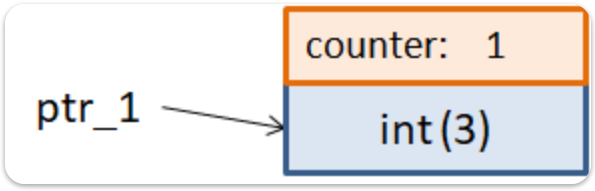
Это их отличает от обычных (классических) указателей языка C/C++.

Контроль памяти указателей типа shared_ptr

Давайте теперь детальнее посмотрим, как указатели типа shared_ptr осуществляют контроль за памятью. Предположим, указатель ptr инициализируется на некоторую область памяти:

```
std::shared_ptr<int> ptr_1 {std::make_shared<int>(3)};
```

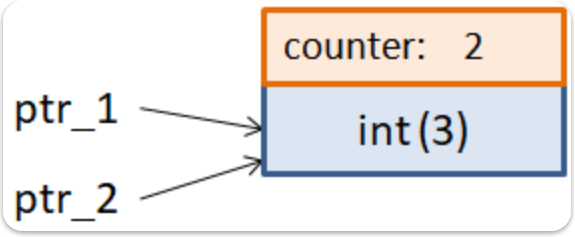
В результате формируется объект самого указателя ptr_1, область памяти для хранения целочисленного значения типа int, а также специальный счетчик counter указателей на выделенную область памяти:



Если объявить еще один указатель на эту же область:

```
std::shared_ptr<int> ptr_2 {ptr_1};
```

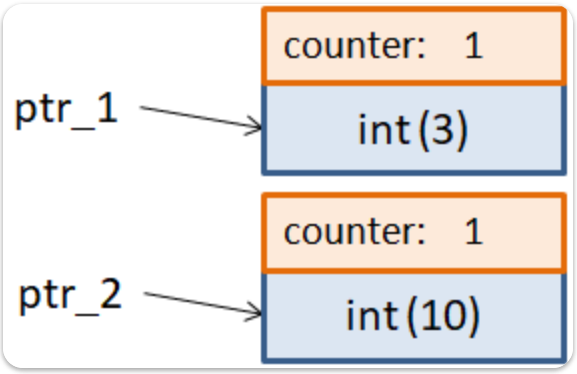
то счетчик counter автоматически увеличивается до двух:



И так далее. С увеличением числа указателей типа shared_ptr на эту область памяти, счетчик counter будет все время увеличиваться.

Предположим теперь, что указатель ptr_2 меняет свой адрес. Например, в результате выполнения такой команды:

```
ptr_2 = std::make_shared<int>(10);
```

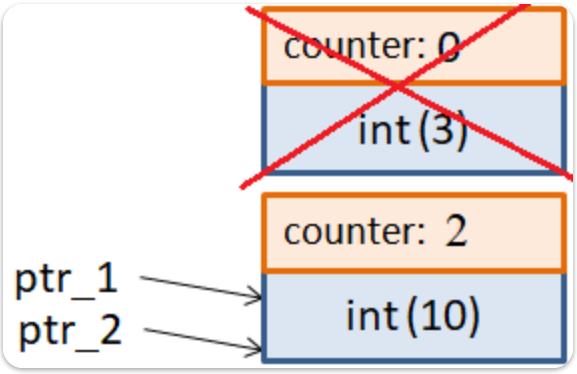


Тогда счетчик для предыдущей области памяти (с int(3)) уменьшится на единицу, а указатель ptr_2 будет ссылаться на другую область памяти со своим счетчиком, у которого значение также будет равно единице.

Далее, если указатель ptr_1 «отвязать» от текущей области памяти:

```
ptr_1 = ptr_2;
```

то счетчик на первую область будет равен нулю, а на вторую – два:



И как только для какой-либо выделенной области счетчик counter устанавливается в ноль, эта область памяти автоматически освобождается. Так реализован контроль за памятью для указателей shared_ptr. Кроме того, смешивать разные типы smart-указателей между собой также запрещено. Например, следующая конструкция приведет к ошибке на этапе компиляции:

```
std::unique_ptr<int> ptr_u {std::make_unique<int>(-1)};
std::shared_ptr<int> ptr_1 {ptr_u};  // ошибка
```

Или конструкция вида:

```
ptr_1 = ptr_u; // ошибка
```

также приведет к ошибке. То есть, мы можем работать либо с указателями unique_ptr, либо с указателями shared_ptr, не смешивая их.

Методы указателей shared_ptr

Следующим шагом рассмотрим методы, которые имеются у smart-указателей типа shared_ptr. Основные из них следующие:

- get() – получение «сырого» указателя на выделенную область памяти;
- reset() – меняет значение указателя на другую область памяти, либо на значение nullptr, если ничего не указано;
- swap() – меняет адреса двух указателей между собой;
- unique() – возвращает true (1), если на выделенную область ссылается только один указатель, и false (0) – в противном случае;
- use_count() – возвращает текущее значение счетчика conter для текущей области памяти.

Давайте посмотрим, как работают эти методы. С первым get() мы уже знакомы по предыдущему smart-указателю. Здесь все то же самое:

```
std::shared_ptr<int> ptr_1 {std::make_shared<int>(3)};
std::shared_ptr<int> ptr_2 {ptr_1};

int* p = ptr_1.get();
```

Следующий метод reset позволяет заменять одну область памяти на другую. Например:

```
    ptr_2.reset(new int[5] {1, 2, 3});
    int* ar = ptr_2.get();

    for(int i = 0;i < 5;++i)
        cout << ar[i] << " ";
```

Обратите внимание, мы выделили новую область памяти с помощью оператора new для массива в 5 элементов. После этого получили стандартный указатель на начало этого массива и с его помощью прочитали и вывели значения в консоль.

К сожалению, указатели типа shared_ptr могут работать с массивами, только начиная с версии языка C++20. До этого придется использовать оператор new.

Оставшиеся методы работают очевидным образом. Например:

```
    ptr_2.reset(new int[5] {1, 2, 3});
    ptr_2.swap(ptr_1);

    cout << *ptr_2 << " " << ptr_1.use_count() << endl;
    cout << ptr_1.unique() << endl;
```

В консоли увидим:

```
3 1
1
```

А если убрать строчку с reset, то вывод будет таким:

```
3 2
0
```

Работа smart-указателей типа shared_ptr с массивами в стандарте C++20

Начиная со стандарта C++20 указатели shared_ptr можно инициализировать и использовать с динамическими массивами. Делается это по аналогии с указателями unique_ptr следующим образом:

```
unsigned total = 10;
std::shared_ptr<int> ar_1 {std::make_shared<int[]>(total)};
```

После этого через указатель ar_1 можно работать как с массивом привычным нам образом:

```
    ar_1[0] = 10;

    for(unsigned i = 0;i < total; ++i)
        cout << ar_1[i] << " ";
```

Заключение

Вот основные возможности «умных» указателей unique_ptr и shared_ptr. Сейчас в практике программирования на C/C++ они крайне рекомендуются при написании программ различного уровня сложности, так как существенно уменьшают проблемы, связанные с утечкой памяти, ее повторным освобождением, доступом к невыделенной памяти и так далее. Поэтому вместо применения операторов new/delete по возможности следует применять тот или иной вид smart-указателей.